A Succinct 3D Visibility Skeleton

Sylvain Lazard, Christophe Weibel, Sue Whitesides, Linqiao Zhang

Discrete Mathematics, Algorithms and Applications

A SUCCINCT 3D VISIBILITY SKELETON

Sylvain Lazard
INRIA Nancy Grand Est, LORIA, Nancy, France.
lazard@loria.fr

Christophe Weibel
McGill University, Math Department, Montreal, QC H3A 2K6, Canada.
weibel@math.mcgill.ca

Sue Whitesides
University of Victoria, Department of Computer Science, B. C. V8W 3P6, Canada.
sue@uvic.ca

Linqiao Zhang
McGill University, School of Computer Science, Montreal, QC H3A 2A7, Canada.
lzhang15@cs.mcgill.ca

Received Day Month Year
Accepted Day Month Year

The 3D visibility skeleton is a data structure that encodes the global visibility information of a set of 3D objects. While it is useful in answering global visibility queries, its large size often limits its practical use. In this paper, we address this issue by proposing a subset of the visibility skeleton, which is empirically about 25% to 50% of the whole set. We show that the rest of the data structure can be recovered from the subset as needed, partially or completely. The running time complexity, which we analyze in terms of output size, is efficient. We also prove that the subset is minimal in the sense that the complexity bound ceases to hold if the subset is restricted further.

Keywords: data structure; 3D visibility; polyhedral scene; computational geometry.

Mathematics Subject Classification 2000: 68U05

1. Introduction

Problems of 3D visibility arise commonly in areas such as computer graphics, computer vision, and robotics. Typical problems include determining the objects in a scene that are visible from a given view point, computing pairwise visibility information among objects, and computing the boundaries of umbra and penumbra cast

by objects under various lighting conditions.[a] These visibility problems can roughly be classified into point-based or surface-based, depending on the nature of the view points or light source.

Point-based visibility problems have been well studied and understood. On the other hand, little is known about surface-based visibility problems. A common approach in practice is to discretize such problems, transforming them into a set of point-based problems, so as to obtain approximate solutions, which is often unsatisfying and time consuming. A method for providing analytical solutions to surface-based visibility problems would be desirable.

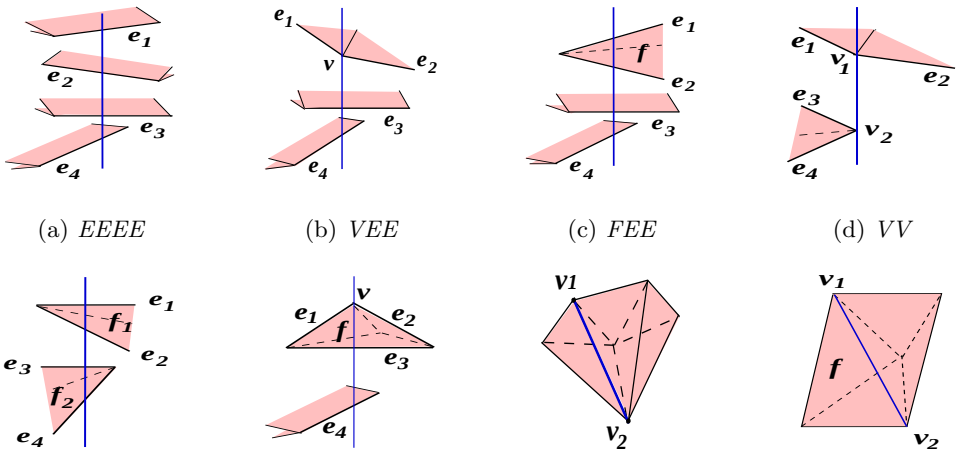
The *visibility complex* is a data structure that is designed to encode global visibility information. This data structure partitions the space of line segments into sets of components, where each component consists of a set of maximal free line segments that share the same occluders, are tangent to the same objects in the scene, and form one connected component. This data structure was initially proposed in 2D by Pocchiola and Vegter [23]. Durand *et al.* extended the visibility complex data structure to 3D and furthermore introduced the 3D visibility skeleton, which essentially consists of the 0D and 1D cells of the visibility complex [11,13], together with their adjacencies. Thus the 3D visibility skeleton is a simpler, smaller data structure than the full visibility complex. It has the structure of a graph, where vertices represent 0D cells and arcs represent 1D cells.

The visibility skeleton has been applied to global illumination computation [9, 11,12], and has resulted in images with high quality shadow boundaries. On the other hand, the applications of [9,11,12] also show that this data structure, although simplified, can still be enormous and thus has limited practical use.

Various other definitions have been proposed for the 3D visibility skeleton [6, 11,19]. These definitions depend on the notion of visual events, which are based on the context (e.g. smooth manifolds, polyhedral scenes) and the intended use of the data structure (e.g. shadow boundary computation). In this article, we refer to a visibility skeleton based on visual events surfaces for a polytope scene, as defined in Demouth *et al.* [5,7]

A visibility skeleton thus defined, when considering input consisting of a set of convex disjoint polytopes, consists of arcs of so-called type *EEE*, and a subset of arcs of so-called type *VE*, and vertices of so-called types *EEEE*, *VEE*, *FEE*, and a subset of vertices of so-called type *VV* (see Figures 1 and 2). For instance, a vertex of type EEEE represents a maximal free line segment that is tangent to four polytopes along four of their edges, and similarly for the other types. Here, the lines supporting the maximal free line segments are required to be tangent to their associated polytopes. Moreover, we only consider among the VV vertices those that correspond to maximal free line segments that lie in some plane tangent to both associated polytopes. This data structure is only a subset of the 3D visibility

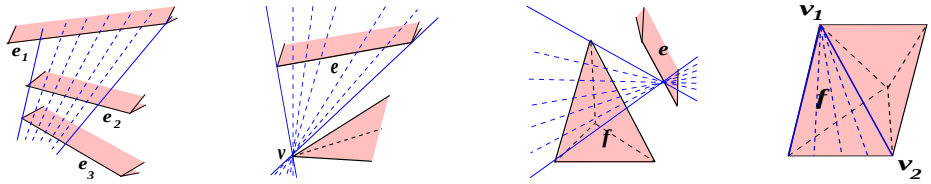
[a]We refer to penumbra and umbra as to regions from which part (but not all) and none of the light source is visible.



(a) *EEEE* (b) *VEE* (c) *FEE* (d) *VV*

(e) *FF* (f) *FvE* (g) *FE* (h) *FVV*

Fig. 1. The eight types of vertices of the 3D visibility skeleton.



(a) *EEE* (b) *VE* (c) *FE* (d) *FVE*

Fig. 2. The four types of arcs of the 3D visibility skeleton.

skeleton defined by Durand *et al.* [10,11], which consists of arcs of type *EEE*, *VE* and *FE*, and vertices of types *EEEE*, *VEE*, *FEE*, *VV*, *FvE*, *FF*, *FE* and *FVV*.[b] For convenience, in this paper, we refer to the former definition as a *succinct 3D visibility skeleton*, and to the latter one as a *full 3D visibility skeleton*. Moreover, we define the vertices contained in the succinct skeleton to be *primary vertices*, and the remaining vertices of the full skeleton to be *secondary vertices*.

From the study of Demouth *et al.*, the size of the succinct 3D visibility skeleton is only about 25% to 50% of the full one [5,7]. However, the skeleton vertices and arcs it contains are sufficient to compute the direct shadow boundaries cast by polytopes. While compact and useful on its own, the succinct 3D visibility skeleton

[b]In [10,11], Durand *et al.* do not assume any structure in the triangles composing a scene. The lines supporting maximal free line segments are always tangent to their associated triangles, but not necessarily to the polyhedra they are part of. In this paper, we consider the triangles to be structured in polytopes and we require tangency between these lines and their associated polytopes.

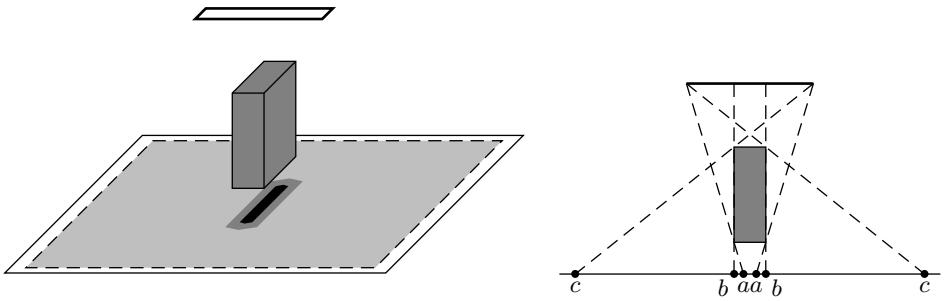


Fig. 3. Scene representing a box in a room with a rectangular light source on the ceiling. The black and white regions represent the umbra and unshaded regions. The union of the light and dark grey regions corresponds to the penumbra. The dark gray shape represents a portion of the penumbra limited by the trace of *FE* arcs. In this region, the visible portion of the light source does not exceed about 40%. The schema on the right represents a section through the middle of the scene. The points a are at the boundary of umbra, and the points c are at the boundary of the unshaded region. The points b are on the maximal free line segments corresponding to an arc *FE* involving a face of the blocker. From a to b, the percentage of the light source that is visible increases linearly from 0% to about 40%, and from b to c, it increases linearly from about 40% to 100%. Since the light grey region can be made arbitrarily large by moving the light source closer to the blocker, the trace of the *FE* arcs on the floor corresponds to a discontinuity of the derivative in the percentage of visible area of the light source.

does not always contain the necessary visibility information for answering global visibility queries. For example, when generating high quality shadows, the typical approach of linearly interpolating light intensity within the penumbrae [2,17,21] is not always sufficient to express the subtlety of penumbrae cast by polytope features, as shown in Figure 3; in particular, arcs *FE* may be needed even though they do not correspond to visual events. Apart from global illumination, other problems such as visibility culling [24], architectural acoustics [14], or endoscopy [18] also need global visibility information.

The full visibility skeleton, on the other hand, contains all the necessary information for most visibility queries. However its large size has been seen as an impediment for its practical use [10,11]. In this paper, we study in detail the 3D visibility skeletons computed from a set of convex disjoint polytopes in general position (defined in Section 2). We prove that knowing the succinct 3D visibility skeleton is sufficient to compute efficiently the secondary vertices; in particular, these computations can be local, that is, only the vertices and arcs of interest need to be computed. Furthermore, the full skeleton can be computed if necessary.

In terms of computing the full skeleton, we prove that, given k disjoint convex polytopes[c] satisfying general position assumptions, with n edges in total, the full visibility skeleton can be computed from the succinct one in $O(p \log p + s \log s)$ time, where p is the number of the primary vertices minus the *EEEE* vertices, and s is

[c]Note that polytopes are bounded polyhedra.

the number of secondary vertices. The worst-case size complexity of the primary vertices of interest to us, *EEEE* vertices excluded, is $\Theta(n^2k)$, and the worst-case size complexity of the secondary vertices is $\Theta(n^2)$ [3]. Thus, in the worst case, $O(p \log p + s \log s)$ is equivalent to $O(n^2k \log n)$.

There exist various algorithms for computing the secondary vertices. For instance one can use the sweep algorithm described in [3,16], or one can also compute, in a brute force way, the possibly occluded candidate secondary vertices and perform ray shooting to check for occlusion [1,8,22]. To our knowledge, the best worst-case running time is $O(n^2k \log n)$, obtained by computing $O(n^2)$ candidate secondary vertices in a brute force way and checking for occlusion using the Dobkin-Kirkpatrick hierarchical representation [8,20], which leads to performing $O(n^2)$ ray shooting queries on each of the k polytopes in $O(\log n)$ time each. Comparatively, the method we propose has the same complexity in the worst case. However, our method is output-dependent, and thus, it can be much more efficient than previously existing algorithms. In addition, our method takes as input the primary vertices, whose observed size is, in a random setting, $Ck\sqrt{nk}$ for a small constant C (see [25]), which is much smaller than the worst-case size, that is $\Theta(n^2k)$.

The rest of this paper is organized as follows. We provide necessary definitions in Section 2. We then introduce the computational relations among the types of visibility skeleton vertices in Section 3. We prove that we can recover the full skeleton from the succinct one in $O(p \log p + m \log m)$ time in Section 4. By a series of examples, we show in Section 5 that none of the primary vertex types can be omitted while maintaining the validity of this result. We finally conclude in Section 6.

2. Preliminaries

We introduce the basic definitions we need in this section. We start with some preliminary definitions in order to explain the types of vertices and arcs of the 3D visibility skeleton of a set of k convex disjoint polytopes in general position. By *general position*, we mean that no edges of different polytopes are parallel, no four points on more than one polytope are coplanar, no line is tangent to more than four polytopes, and no four edges are on a hyperboloid quadratic surface [4].

A *support vertex* of a line is a polytope vertex that lies on the line. A *support edge* of a line is a polytope edge that intersects the line but has no endpoint on it (a support edge intersects the line at only one point of its relative interior). A *support* of a line is one of its support vertices or support edges. A maximal free line segment is a maximal subset of a line that does not intersect the interior of any polytope. The supports of a segment are defined to be the supports of the relative interior of the segment; thus if a maximal free line segment ends at a vertex of a polytope, this vertex is not a support. A *support polytope* of a line is a polytope that a support of the line lies on.

Full 3D visibility skeleton. We introduce this data structure based on the work of Durand *et al.* [10,11], and often refer it as the full skeleton. The 3D visibility skeleton is a graph that consists of vertices and arcs. A skeleton vertex is a point in the space of maximal free line segments, and a skeleton arc is a connected sequence of points in the same space, with a skeleton vertex at each extremity. (See Figure 1 and 2 for graphical illustrations, and [10,11] for more details).

Full 3D visibility skeleton vertices. There are eight types of skeleton vertices, shown in Figure 1. Note that unless stated otherwise, no two supports come from the same polytope. A skeleton vertex has type *EEEE* if its set of supports consists of four edges; *VEE* if its set of supports consists of a vertex and two edges; *FEE* if its set of supports consists of two edges on one face, and two additional edges; *VV* if its set of supports consists of two vertices; *FF* if its set of supports consists of two edges on one face, and two edges on another face; *FvE* if its set of supports consists of a vertex and an edge on one face, and an edge; *FE* if its set of supports consists of two adjacent vertices of the same polytope; and *FVV* if its set of supports consists of two non-adjacent vertices on the same face of a polytope.

We also consider a special case of *FvE* vertex called an *extremal FvE* vertex, as shown in Figure 4. An extremal *FvE* vertex is also in the plane containing a face incident to the polytope vertex, but does not intersect that face except on the vertex. Note that our definition is consistent with the discussion in [15, § 5.2.1], which shows that without the extremal *FvE* vertex, some arcs are missing end vertices. The set of supports of extremal *FvE* vertex consists of a vertex and only one additional edge on a different polytope, in contrast, a non-extremal *FvE* vertex has one more support edge.

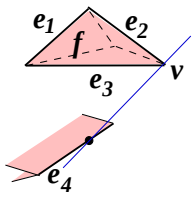


Fig. 4. The extremal case of type *FvE* vertex.

We note that with these definitions, all skeleton arcs have a skeleton vertex at both ends.

It should be stressed that the maximal free line segment corresponding to a skeleton vertex is tangent to all its support polytopes.

Full 3D visibility skeleton arcs. There are four types of skeleton arcs. We define them, based on [10,11], as follows, and show graphical illustrations in Figure 2. Note that unless stated otherwise, no two supports will come from the same polytope. An arc has type: *EEE* if its set of supports consists of three edges; *VE* if its set of supports consists of a vertex and an edge; *FE* if its set of supports consists of two edges on one face, and one additional edge; *FVE* if its set of supports consists of one vertex and one edge on the same face which is not incident to the vertex.

Again, we emphasize that the maximal free line segments corresponding to a skeleton arc are tangent to all their support polytopes.

Succinct 3D visibility skeleton, primary and secondary vertices. We define the *primary vertices* to be the skeleton vertices of types *EEEE*, *VEE*, *FEE*, together with the vertices of type *VV* whose corresponding maximal free line segment lies in a plane tangent to both polytopes; and the *secondary vertices* to be the remaining vertices of type *VV*, and the vertices of types *FF*, *FvE*, *FE*, and *FVV*.

The succinct 3D visibility skeleton is the subgraph of the full 3D visibility skeleton that contains only primary vertices, and skeleton arcs connecting two primary vertices.

Constraints. Recall that for any skeleton vertex, a support vertex is a polytope vertex that lies on the relative interior of the maximal free line segment, and a support edge is a polytope edge that intersects the maximal free line segment in both their relative interiors. For any skeleton vertex, we define its *constraints* to be the polytope edges that intersect the maximal free line segment corresponding to the skeleton vertex, and such that the plane containing the edge and the maximal free line segment is tangent to the polytope containing the edge. In other words, the maximal free line segment corresponding to the skeleton vertex can be perturbed so that it intersects the interior of the polytope edge, while remaining tangent to the polytope. We define constraints similarly for arcs. We note that any support edge is a constraint, and that any support vertex is incident to at least two constraints that are not support edges (in the case of extremal *FvE* vertices, there are at least three constraints, and possibly four). A constraint edge which is not a support edge is called a *salient edge*. We note that any skeleton vertex has, under our general position assumption, at least four constraints, and up to seven in some cases. Any skeleton arc has three.

Remark 2.1. By continuity, the constraints of an arc are all constraints of its end vertices.

3. Computational Relations among the Visibility Skeleton Vertices

Recall from Section 2 that a 3D visibility skeleton vertex corresponds to a maximal free line segment that has 0-degrees of freedom, subject to the condition that the maximal free line segment keeps the same constraints.

Remark 3.1. We specify that the knowledge of a skeleton vertex includes its corresponding maximal free line segment and its constraint edges, that is, any support polytope edges, any support polytope vertices with incident salient edges, and all support polytopes. The knowledge of a polytope includes cyclic orderings of edges around vertices and faces. Computing a skeleton vertex includes computing the maximal free line segment and all these constraint edges, and similarly for skeleton arcs. Furthermore, we create during preprocessing a binary search data structure on the cyclic orderings of edges around each polytope vertex and face. This is done in $O(n \log n)$, where n is the number of polytope edges in the scene.

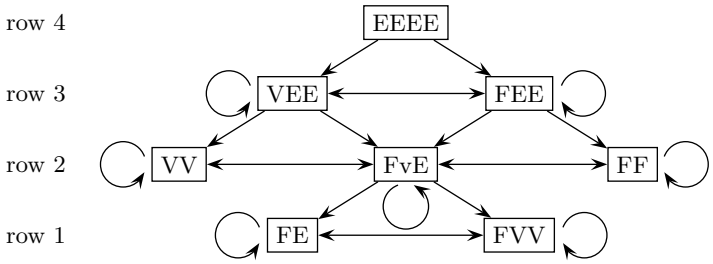


Fig. 5. The possible computational relations among the types of 3D visibility skeleton vertices.

Note that, by Remark 2.1, and since a skeleton vertex includes the knowledge of its constraint edges, we can determine all skeleton arcs that are incident to a skeleton vertex by removing, in turn, each edge from the list of its constraints. Moreover, the computation time is constant.

Given a skeleton arc, which includes the knowledge of its constraints and support polytopes, any incident skeleton vertex can be computed without additional knowledge, provided that the skeleton vertex has the same set of support polytopes as the skeleton arc. The computation involves searching edges incident to a polytope vertex to find salient edges, or edges on a polytope face to find the support edges of the incident skeleton vertex. Thanks to the binary search data structure of Remark 3.1, this requires only $O(\log \delta)$ time computation, where δ is the maximum degree of a polytope vertex or number of edges on a polytope face. For example, given an *EEE* arc, one can compute the vertices of types *VEE* and *FEE* that are incident to the *EEE* arc in $O(\log \delta)$ time. But, if an *EEE* arc is incident to an *EEEE* vertex, then this *EEEE* vertex can not be computed directly from the *EEE* arc, since the *EEEE* vertex has an additional, unknown support polytope.

The possible computational relations among skeleton vertex types are summarized in the diagram in Figure 5. The edges in this diagram give all possible pairs of vertex types that can be connected by an arc of the full visibility skeleton. Furthermore, an arrow oriented from one vertex type to another indicates that the set of support polytopes of a vertex of the former type contains that of a vertex of the latter type. This means that vertices of the latter type can be computed from adjacent vertices of the former type in $O(\log \delta)$ time. Note that skeleton vertices of types appearing in rows 1 through 4 of the diagram in Figure 5 are supported by $1, \ldots, 4$ polytopes, respectively.

We prove the correctness of the computational relations illustrated in the diagram of Figure 5 in Lemmas 3.2 and 3.3 below.

Lemma 3.2. *Let $\mathcal{X}$ and $\mathcal{Y}$ denote vertex types in the full skeleton graph, such that in Figure 5, $\mathcal{X}$ has more support polytopes than $\mathcal{Y}$ (so $\mathcal{X}$ appears in a higher row than $\mathcal{Y}$), and there is an edge directed from $\mathcal{X}$ to $\mathcal{Y}$. Then we cannot, from a skeleton vertex of type $\mathcal{Y}$, determine an adjacent skeleton vertex of type $\mathcal{X}$ in time*

independent of the number of input polytopes, without more knowledge than specified in Remark 3.1.

Proof. Since any vertex of type $\mathcal{X}$ has more support polytopes than any vertex of type $\mathcal{Y}$, when computing an incident vertex from the arc that is created by a vertex of type $\mathcal{Y}$, the additional support polytope, and furthermore the additional support edge (or support vertex) on the additional support polytope, is unknown. This requires a search for the additional polytope and the additional support edge (or support vertex) on the polytope. In general, this search cannot be done directly from a vertex of type $\mathcal{Y}$, given only the knowledge of its supports. □

Lemma 3.3. *If a scene is in general position, then from any skeleton vertex, it is possible to compute each adjacent skeleton vertex in* $O(\log \delta)$ *time, where* δ *is the maximum degree of a polytope vertex or number of edges on a polytope face, provided that all support polytopes of the adjacent skeleton vertex are also support polytopes of the given skeleton vertex and of the connecting skeleton arc.*

Proof. By the assumption of the Lemma, we already know that the support polytopes of the adjacent skeleton vertex are a subset of the given skeleton vertex.

From the starting skeleton vertex, we need to find the incident skeleton arcs. Since we know the constraint edges of the skeleton vertex, and there are at most a constant number of them by Remark 2.1, we can find its incident skeleton arcs in constant time by removing in turn each of these constraints. As the polytopes are in general position, the number of constraints is bounded; there may however be as many as seven of these constraints and it is sometimes necessary to remove more than one of these at the same time.

We now consider how to deal with each type of arc incident to a given vertex. From an *EEE* arc, it is possible to find unknown incident skeleton vertices of type *VEE* or *FEE* by checking all polytope faces and vertices incident to the polytope edges, and computing the corresponding candidates. Since their number is constant, we can find the incident vertices in constant time by enumeration. Similarly, from a *VE* arc, we find unknown incident *VV* vertices by checking polytope vertices incident to the polytope edge. From an *FE* arc, we find unknown *FF* vertices by checking polytope faces incident to the polytope edge.

Moreover, from a *VE* arc, we find unknown incident *FvE* vertices by checking the four polytope faces that are incident to the two salient edges incident to the polytope vertex. From an *FE* arc, we find unknown *FvE* vertices by checking the four polytope vertices that are incident to the two polytope edges lying on the polytope face.

Thus we can find the maximal free line segment that corresponds to an adjacent skeleton vertex in constant time, along with its support vertices or faces.

According to Remark 3.1, we also need to compute the constraint edges. For a *VEE* or *VV* vertex, these are salient edges, which can be computed by checking

through all the polytope edges incident to the support polytope vertices. This can be done in $O(\delta)$ time. For an *FEE* or *FF* vertex, these are support edges, which can be computed by checking all the polytope edges on the support polytope faces. Again, this can be done in $O(\delta)$ time.

Similarly, one can prove the computational relations shown in Figure 5 for skeleton vertices of type *FE* and *FVV*. □

In summary, for any directed edge in Figure 5 from type $\mathcal{X}$ to type $\mathcal{Y}$, Lemma 3.3 shows that it is possible to compute a vertex of type $\mathcal{Y}$ from an adjacent vertex of type $\mathcal{X}$; Lemma 3.2 shows that it is impossible to compute a vertex of type $\mathcal{X}$ from a vertex of type $\mathcal{Y}$ in time independent of the number of polytopes.

4. Recovery of the Full Skeleton

We show in this Section that the full visibility skeleton can be recovered from the succinct one in $O(p \log p + s \log s)$ time, where p is the number of primary vertices minus the *EEEE* vertices, and s is the number of secondary vertices.

Recall that the five types of secondary vertices are *FF*, *FvE*, *FVV*, *FE*, and a subset of *VV*. The vertices of type *FVV* and *FE* are easy to find on their own, and can be computed separately (see Remark 4.8). In this section, we show how to compute vertices of type *FvE*, *FF*, and the subset of *VV* that belongs to the secondary vertices. For this, we explore the subgraph of the full visibility skeleton consisting of *VE* and *FE* arcs and their incident vertices, that is, the *VEE*, *FEE*, *VV*, *FvE* and *FF* vertices. We call this subgraph the *partial graph*.

We first prove that all connected components of the partial graph contain at least a primary vertex of type *VV*, *VEE* or *FEE*. This allows us to find all *FvE*, *FF* and the remaining *VV* vertices by simple graph exploration, examining vertices adjacent to those we have already computed.

To prove that all connected components contain a primary vertex of type *VV*, *VEE* or *FEE*, we proceed as follows. If a pair P_i, P_j of polytopes supports a skeleton vertex, we define G_{ij} to be the subgraph consisting of all skeleton vertices and arcs that have P_i and P_j as supports. We show in the following theorem that each connected component of each subgraph G_{ij} contains a primary vertex of the specified types. This allows us to enumerate each subgraph separately starting from these primary vertices.

Theorem 4.1. *Let P_i and P_j be a pair of polytopes that supports a skeleton vertex. Then each connected component of the subgraph G_{ij} defined as above contains a primary vertex.*

Before we provide the details of the proof of Theorem 4.1, we sketch a general outline of it. For each pair of polytopes P_i, P_j that support at least one skeleton vertex, we define an *objective function*, which associates a value to each maximal free line segment tangent to both polytopes. Recall that the graph G_{ij} is defined

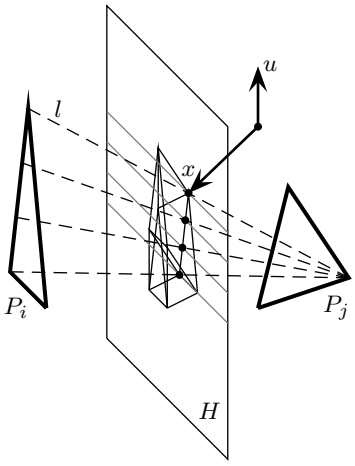


Fig. 6. The figure on plane H represents the intersections with H of the maximal free line segments corresponding to the skeleton arcs and vertices supported by P_i and P_j. The value of each of the maximal free line segments is defined by a linear function of its intersection with the plane H.

to be the subgraph consisting of all skeleton vertices and arcs that have P_i and P_j as supports. We prove that each local minimum of the objective function on each connected component of the G_{ij} graph is a primary vertex.

For each pair of polytopes P_i and P_j that support at least one skeleton vertex, we define the following objective function (See Figure 6). Let H be a plane separating P_i and P_j, and let u be a vector that is in *generic* direction (explained below). For any maximal free line segment l tangent to P_i and P_j that corresponds to a skeleton vertex or a point of a skeleton arc, we define the *value* of l to be $f(l) = u \cdot x$, where x is a vector representing the intersection of H with the supporting line of l.[d] Note that l always intersects H, being tangent to P_1 and P_2, which are separated by H.

Lemma 4.2. *Any vertex v of the visibility skeleton of type* FF *or* FvE *that is supported by P_i and P_j has an adjacent vertex v', connected by a* VE *or* FE *arc, such that the value of f along the arc continuously decreases from v to v'.*

Proof. If maximal free line segments are all tangent to the same edge and all intersect at a common single point off the edge, their intersections with plane H lie on a straight line, the intersection of H with the plane containing the tangent edge and the common point. We can parametrize these free line segments by $l(t)$, $t \in [0, 1]$, such that $l(0)$ and $l(1)$ correspond to the maximal free line segments tangent to the extremities of the edge, and $f(l(t))$ is an affine function, which is not constant since u is in generic position, so $f(l(t))$ has a minimum at 0 or 1.

[d]We can define the generic direction of u to be $u = (1, \varepsilon, \varepsilon^2)$, so that when ε goes to 0, $f(l_1) \neq f(l_2)$, for any maximal free line segments l_1 and l_2 corresponding to adjacent skeleton vertices.

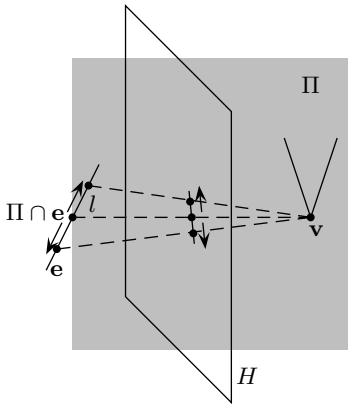


Fig. 7. The intersections with H of maximal free line segments on the two *VE* arcs on each side of an extremal *FvE* vertex move in opposite directions.

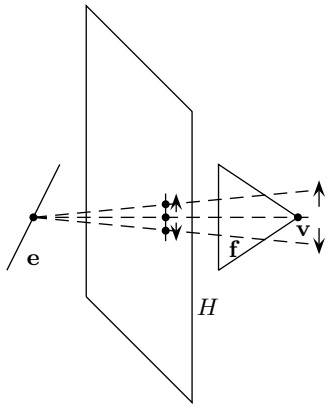


Fig. 8. The intersections with H of maximal free line segments on the two *FE* arcs on each side of a non-extremal *FvE* vertex move in opposite directions.

In particular, the set of maximal free line segments corresponding to a *VE* or *FE* arc of the visibility skeleton share a 3D point and are tangent to a common polytope edge e. Therefore, the value of a maximal free line segment increases or decreases monotonically between two skeleton vertices connected by arcs of these types.

Recall that extremal *FvE* vertices correspond to maximal free line segments l tangent to a polytope edge and a vertex of a different polytope, and are in the supporting plane Π of a face incident to the vertex, without intersecting that face except at the vertex (Figure 7). An extremal *FvE* vertex always has two incident *VE* arcs corresponding to the same polytope vertex and polytope edge, rotating about the polytope vertex from l in opposite directions. Therefore, the maximal free line segment corresponding to an extremal *FvE* vertex is in the middle of a set of free line segments defined by a vertex and an edge, which corresponds to two incident skeleton arcs. Since it is in the middle, its value is not minimum, so the value is decreasing along one of the arcs.

Non-extremal *FvE* vertices correspond to maximal free line segments tangent to a polytope face and a polytope vertex of that face, and to an edge of a different polytope (Figure 8). If a non-extremal *FvE* vertex is defined by an edge **e** of polytope P, a vertex **v** and a face **f** of polytope P', then it has two incident *FE* arcs corresponding to the same intersection point on **e** and polytope edge (incident to **f**, but not **v**), whose corresponding maximal free line segments rotate about **e** in opposite directions. Again, the corresponding maximal free line segment is in the middle of a set of free line segments defined by a vertex and an edge, which corresponds to two incident skeleton arcs, and the value is decreasing along one of the arcs.

Vertices of type *FF* have four incident *FE* arcs. Let p and p' denote the intersection points of the maximal free line segment corresponding to an *FF* vertex with the two support edges on one of the two polytopes. Then the two *FE* arcs, obtained by rotating the maximal free line segment corresponding to the *FF* vertex around p and p', will move in opposite directions on a line in H. Therefore, one of them is decreasing. □

Lemma 4.3. *Let v be a vertex of the visibility skeleton of type* VV*. If all adjacent vertices of v have a higher value than v, then v lies in a plane tangent to both polytopes, i.e., v is a primary vertex.*

Proof. Let v be a *VV* vertex supported by the polytope vertices $\mathbf{v}$ and $\mathbf{v'}$ in the polytopes P and P', respectively, such that all adjacent skeleton vertices have a higher value. Let $\mathbf{x}$ be the intersection of v with H. Without loss of generality, we assume $\mathbf{x}$ to be the origin. Since H contains the origin, its equation is $H = \{x : a \cdot x = 0\}$ for some vector a. We modify u into u', such that u' is perpendicular to v, but the value function does not change. This can be done by setting

$$u' = u - \frac{u \cdot d}{a \cdot d} a,$$

where d is a vector from $\mathbf{v}$ to $\mathbf{v'}$. It is easy to check that $u' \cdot d = 0$ and $u \cdot x = u' \cdot x$ for any x in H. Let K denote the plane through x perpendicular to u'.

Vertices of type *VV* have four incident *VE* arcs, two supported by each polytope vertex. Let $\mathbf{e_1}$ and $\mathbf{e_2}$ be the two salient edges of v incident to $\mathbf{v'}$; then $\mathbf{v}$ and $\mathbf{e_1}$ generate a *VE* arc incident to v, as well as $\mathbf{v}$ and $\mathbf{e_2}$. Viewed from $\mathbf{v}$, the silhouette of P projected on H is contained in the cone between $\mathbf{e_1}$ and $\mathbf{e_2}$ (see Figure 9). By assumption, the value of maximal free line segments is increasing along the four incident *VE* arcs, so P is in the half-space $u' \cdot x \geq 0$.

Similarly, P' is in the half-space $u' \cdot x \geq 0$, and so v lies in the plane K defined by $u' \cdot x = 0$ which is tangent to P and P'. □

We are now ready to prove Theorem 4.1. Let us recall the statement:

Theorem 4.1 *Let P_i and P_j be a pair of polytopes that supports a skeleton vertex. Then each connected component of the subgraph G_{ij} defined as above contains a primary vertex.*

Proof. Since any non-empty subgraph G_{ij} is finite, it has a minimum. From Lemma 4.2, we know that vertices of type *FF* or *FvE* cannot be local minima. From Lemma 4.3, we know that any vertex of type *VV* that is a local minimum is a primary vertex. It follows that the minimum of G_{ij} is a primary vertex. □

We now show how to explore the partial graph, in order to compute the secondary vertices of types *VV*, *FvE* and *FF* from primary vertices of types *VEE*, *FEE*

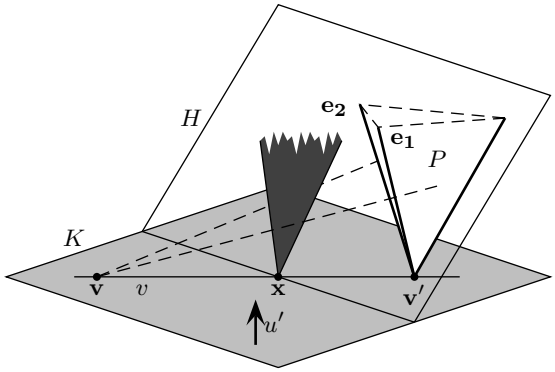


Fig. 9. The silhouette of the polytope P from $\mathbf{v}$ projected on H is inside the cone of the projected salient edges. If the salient edges are in the half-plane $u' \cdot x \geq 0$, so is the polytope.

and *VV*. To explore the partial graph efficiently, we compute the secondary vertices of type *VV*, *FvE* or *FF* that occur in a certain sequence of *VE* or *FE* arcs. In what follows, we will show that this can be done in time $O(p' \log p + s')$, where p' and s' are the number, in the considered sequence, of primary and secondary vertices respectively, and p is the total number of primary vertices that are not *EEEE* in the succinct visibility skeleton.

We consider *VE* arcs and *FE* arcs separately. We define a *sequence of VE arcs* to be a maximal path of *VE* arcs that all share the same support edge and support vertex. We define a *sequence of FE arcs* to be a maximal path of *FE* arcs that share the same support polytope face, and that are tangent to the same other polytope. Note that our definition does not require a sequence of *FE* arcs to be supported by the same edge of the other polytope. The position in these sequences of skeleton vertices of different types is explained in the two following lemmas. The proofs are a simple application of the properties of skeleton vertices.

Lemma 4.4. *Any sequence of* VE *arcs has a* VV *vertex, a* VEE *vertex, or a non-extremal* FvE *vertex at each extremity, and arcs in the sequence are separated by extremal* FvE *vertices or* VEE *vertices.*

Proof. Any non-extremal *FvE* vertex has a single incident *VE* arc. Any *VV* vertex has four of them, but no two of them are supported by the same polytope vertex and edge. Thus, any non-extremal *FvE* or *VV* vertex is an extremity of a sequence of *VE* arcs. Any extremal *FvE* vertex has two incident *VE* arcs supported by the same polytope vertex and edge, so it is in the middle of a sequence. A *VEE* vertex has three or four incident *VE* arcs, which are supported by two different polytope edges. When two arcs are supported by the same edge, the skeleton vertex is in the middle of the sequence; when only one arc is supported by an edge, the vertex is an extremity of the sequence. □

Lemma 4.5. *Any sequence of* FE *arcs has an extremal* FvE *vertex or* FEE *vertex at each extremity, and arcs in the sequence are separated by non-extremal* FvE *vertices,* FEE *vertices or* FF *vertices.*

Proof. Any extremal *FvE* vertex has a single incident *FE* arc. It is therefore an extremity of a sequence of *FE* arcs. Any non-extremal *FvE* vertex has two incident *FE* arcs supported by the same polytope face and polytope edge. It is in the middle of a sequence. And any *FF* vertex has four incident *FE* arcs, each polytope face supporting two of them, which are tangent to different edges on the other polytope. They are therefore in the middle of a sequence. An *FEE* vertex has three or four incident *FE* arcs, which are supported by two different polytope edges. When two arcs are supported by the same edge, the vertex is in the middle of the sequence; when only one arc is supported by an edge, the vertex is an extremity of the sequence. □

The next two lemmas state that any sequence of arcs can be completely explored starting from its primary vertices. Recall that p is the total number of primary vertices in the succinct visibility skeleton minus the *EEEE* vertices, and p' and s' are the number of primary, respectively secondary vertices in a sequence of arcs.

Lemma 4.6. *Any sequence of* VE *arcs can be computed in* $O(p' \log p + s' + \log \delta)$ *time if we know all the primary vertices, or, when the sequence contains no primary vertex, if we know one secondary vertex.*

Proof. First, we find the primary vertices in the sequence from the total list of primary vertices in the succinct visibility skeleton. If the list is sorted by supports, this can be done in $O(p' \log p)$. We then show that secondary vertices can be computed in linear time in their number.

All skeleton vertices in a sequence of *VE* arcs are supported by a vertex $\mathbf{v}$ on polytope P and an edge $\mathbf{e}$. Suppose $\mathbf{x}$ is a point moving on $\mathbf{e}$, and let us consider the maximal free line segment l containing $\mathbf{x}$ and $\mathbf{v}$. The constraints of l are $\mathbf{e}$ and two salient edges incident to v, denoted $\mathbf{e_1}$ and $\mathbf{e_2}$. Let p_1, p_2 be the two planes containing $\mathbf{x}$ and $\mathbf{e_1}$, $\mathbf{e_2}$ respectively. Then plane p_1 (respectively p_2) is tangent to P and contains $\mathbf{x}$, $\mathbf{v}$ and $\mathbf{e_1}$ (respectively $\mathbf{e_2}$). As $\mathbf{x}$ moves along $\mathbf{e}$, planes p_1 and p_2 roll around the faces and edges incident to $\mathbf{v}$. Let C be the polyhedral cone created by faces incident to $\mathbf{v}$, and let C' be the centrally symmetric cone also having its apex at $\mathbf{v}$. If the supporting line of $\mathbf{e}$ does not intersect C or C', the two planes roll in the same direction around $\mathbf{v}$ (Figure 10). Otherwise, they roll in opposite directions (Figures 11 and 12).

Suppose we know one of the extremities of the sequence of *VE* arcs, which can be either a *VV*, a *VEE* or a non-extremal *FvE* vertex. We examine each of the cases in turn as follows.

Case i): *The extremity is of type* VV (supported by $\mathbf{v}$ and $\mathbf{v}'$): Then the planes tangent to P contain its salient edges, $\mathbf{e_1}$ and $\mathbf{e_2}$; the supporting planes of the polytope faces incident to $\mathbf{e_1}$ or $\mathbf{e_2}$ intersect the supporting line of the polytope

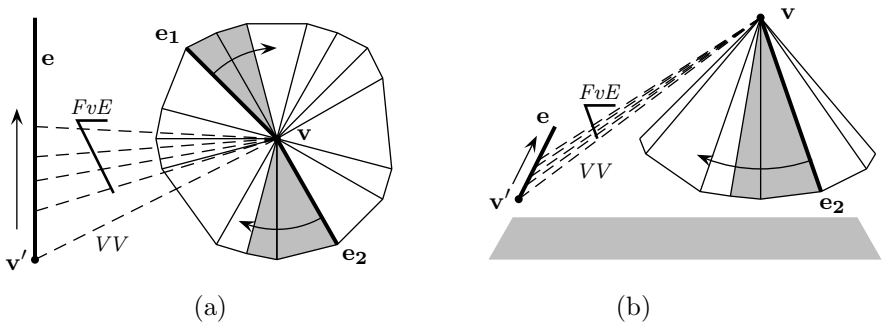


Fig. 10. (a) Bird's eye view and (b) 3D view of extremal *FvE* vertices whose supports are polytope edge $\mathbf{e}$ and two sequences of faces incident to $\mathbf{v}$, starting from $\mathbf{e_1}$ and $\mathbf{e_2}$, which create *VE* arcs with $\mathbf{v}'$.

edge $\mathbf{e}$ on either side of $\mathbf{v}'$ (see Figures 10 and 11). From those two edges ($\mathbf{e_1}$ and $\mathbf{e_2}$), circling around the polytope vertex $\mathbf{v}$, we enumerate the two sequences of faces incident to $\mathbf{v}$. Their supporting planes intersect the polytope edge $\mathbf{e}$, and each of the intersections corresponds to an extremal *FvE* vertex.

If there is a *VEE* vertex in the sequence of arcs, then we already know the vertex, as well as the point where the maximal free line segment corresponding to the *VEE* vertex intersects the edge $\mathbf{e}$. Thus we can insert it in the sequence when the intersections reach that point. The *VEE* vertex can end the sequence; otherwise we keep enumerating the faces incident to $\mathbf{v}$.

If the sequence of arcs ends with a *VV* vertex, we stop when the intersections reach the other end of edge $\mathbf{e}$.

In case the two sequences of faces are turning in opposite directions, and they turn until they meet (see Figure 11), this indicates a non-extremal *FvE* vertex at the end of the sequence of arcs.

Case ii): *The extremity of the sequence of* VE *arcs formed by vertex* $\mathbf{v}$ *and edge* $\mathbf{e}$ *is a* VEE *vertex* (supported by $\mathbf{v}$, $\mathbf{e}$ and $\mathbf{e}'$): then let $\mathbf{e_1}$ and $\mathbf{e_2}$ denote the salient edges of the *VEE* vertex, incident to $\mathbf{v}$, and we continue as in the case that the extremity is of type *VV*.

Case iii): *The extremity of the sequence of* VE *arcs formed by vertex* $\mathbf{v}$ *and edge* $\mathbf{e}$ *is a non-extremal* FvE *vertex* (supported by $\mathbf{f}$, $\mathbf{v}$ and $\mathbf{e}$): Then we enumerate from $\mathbf{f}$ the two sequences of faces incident to $\mathbf{v}$ and proceed as above.

In case we do not know an extremity of the sequence of arcs, but we do know a vertex in the middle, we can explore the sequence in each direction using the same method, since we know the constraints of the vertex. If it is of type *VEE*, then let $\mathbf{e_1}$ and $\mathbf{e_2}$ denote its salient edges incident to $\mathbf{v}$, and continue as above. If it is of type extremal *FvE* (supported by $\mathbf{f}$, $\mathbf{v}$ and $\mathbf{e}$), then let $\mathbf{e_1}$, $\mathbf{e_1'}$ and $\mathbf{e_2}$ be three salient edges incident to $\mathbf{v}$, with $\mathbf{e_1}$ and $\mathbf{e_1'}$ incident to $\mathbf{f}$. We then enumerate faces incident to $\mathbf{v}$ in one direction starting with $\mathbf{e_1}$ and $\mathbf{e_2}$, and in the other direction starting with $\mathbf{e_1'}$ and $\mathbf{e_2}$.

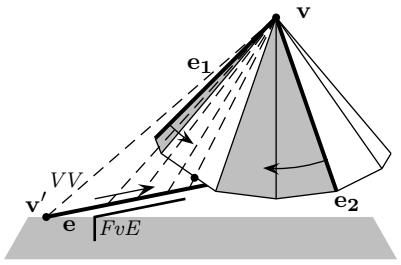


Fig. 11. When the polytope edge **e** intersects with the polyhedral cone of the faces incident to **v**, the two sequences of faces that are supports of extremal *FvE* vertices turn in opposite directions until they meet, which indicates a non-extremal *FvE* vertex.

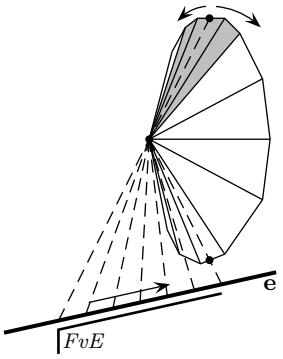


Fig. 12. In some configurations, a sequence of *VE* arcs may contain extremal *FvE* vertices only, with a non-extremal *FvE* vertex at each end.

Note that if the sequence does not have a *VV* or *VEE* vertex at either extremity, then the supporting planes of *all* faces incident to the vertex **v** intersect with edge **e** (see Figure 12). All of these intersections correspond to *FvE* vertices.

Finally, if an extremity of a sequence is a non-extremal *FvE* vertex, we need to find its constraints among the edges incident to the support polytope face, and this can be done in $O(\log \delta)$ time. □

Lemma 4.7. *Any sequence of* FE *arcs can be computed in* $O(p' \log p + s' + \log \delta)$ *if we know all the primary vertices, or, when the sequence contains no primary vertex, if we know one secondary vertex.*

Proof. As in Lemma 4.6, we first find the primary vertices in the sequence from the complete list of primary vertices in the succinct visibility skeleton in $O(p' \log p)$ time. We then show that secondary vertices can be computed in linear time in their number.

If the extremity of the sequence of *FE* arcs formed by face **f** and polytope P is an *FEE* vertex (supported by face **f**, edge **e** on P and $\mathbf{e}'$ on some other polytope), then let $\mathbf{e_f}$ and $\mathbf{e'_f}$ denote its two support edges on face **f**. From $\mathbf{e_f}$ or $\mathbf{e'_f}$, circling around the face **f**, the two sequences of polytope vertices (on face **f**) will create a sequence of arcs joined by non-extremal *FvE* vertices with **e**.

If there is an *FEE* in the sequence of arcs, then we know the vertex, as well as its support edges on **f**. If the sequence of arcs reaches an *FEE* vertex, we insert the vertex into the sequence of arcs. The *FEE* vertex can end the sequence; otherwise we keep enumerating the vertices of **f**.

If the sequence of arcs contains an *FF* vertex, it corresponds to face **f** and a face $\mathbf{f}'$ that is incident to edge **e**. These are inserted in the sequence of arcs when the

sequences of vertices cross the supporting plane of $\mathbf{f}'$.

In case we do not know an extremity of the sequence of arcs, then they are the type of extremal *FvE*. In this case, all vertices of face $\mathbf{f}$ create an extremal or non-extremal *FvE* vertex with the other polytope. So starting from any skeleton vertex we know, we can enumerate polytope vertices on face $\mathbf{f}$, adding vertices of type *FF* and *FEE* along the way as above.

Finally, if an extremity of a sequence is an extremal *FvE* vertex, we need to find its constraints among the edges incident to its support vertex, and this can be done in $O(\log \delta)$ time. □

If an extremity of a sequence is a secondary vertex, we need to find the constraints of that vertex among the edges incident to a polytope vertex or to a face, and this can be done in $O(\log \delta)$ time for each sequence, using the binary search data structure from Remark 3.1.

We describe now our exploration procedure in detail. We compute the secondary vertices by exploring the (unknown) partial graph. That is, we first examine each primary vertex of type *VV*, *VEE* or *FEE*, and find all secondary vertices of type *VV*, *FvE* and *FF* on adjacent sequences of arcs. We keep a list of discovered secondary vertices, and check before adding any new one whether it is already there. We then examine recursively each vertex in that list, looking again for secondary vertices on adjacent sequences of arcs, which are added to the end of the list on the condition that they are not yet there. In this sense, we are treating the list like a queue. To search the list efficiently, we order it (lexicographically for example), and keep track of the queuing order by adding to each vertex a pointer to the next one to be examined. Checking whether a vertex is already in the list is then done in logarithmic time.

Since any vertex is adjacent to a constant number of arcs, the search for secondary vertices can be done in $O(p \log p + s'(\log s' + \log \delta))$ time. Each of the s' secondary vertices found during the search is computed in $O(\log \delta)$ time and added to the list of secondary vertices in $O(\log s')$ time.

Remark 4.8. The *FE* vertices correspond to edges of polytopes and can be computed by simple enumeration. Furthermore, *FVV* vertices correspond to diagonals of faces of polytopes, and can also be found by simple enumeration.

Theorem 4.1 and Lemmas 4.6 and 4.7 yield the following result:

Theorem 4.9. *Given the succinct visibility skeleton, one can compute the full visibility skeleton from the succinct one. This computation can be done in* $O(p \log p + s \log s)$ *time, where* p *is the number of primary vertices of type* VV*,* VEE *and* FEE*, and* s *is the number of secondary vertices (of type* VV*,* FvE*,* FF*,* FE*, and* FVV*).*

Proof. We start with the knowledge of all the primary vertices, and find the secondary vertices. In order to do that, we explore the partial subgraph of the visibility

skeleton containing all *VE* and *FE* skeleton arcs and the skeleton vertices at their extremities. We find in this way all vertices of type *VV*, *FvE* and *FF*.

By Theorem 4.1, we know at least one vertex in each connected component of this partial subgraph. Lemmas 4.6 and 4.7 show that from any vertex in a sequence of *VE* or *FE* arcs, we can find all the secondary vertices in the sequence in time linear in their number. As any unknown vertex is connected to a known vertex through a series of sequences of arcs, we can find all vertices.

We have seen that using our special exploration procedure, a graph of p known vertices and s' unknown vertices can be explored in $O(p \log p + s'(\log s' + \log \delta))$ time. Vertices of type *FE* and *FVV* are computed separately in linear time in their number s''. Thus the complete enumeration is done in $O(p \log p + s'(\log s' + \log \delta) + s'')$ time, where $s' + s'' = s$. Note that δ is in $O(s'')$, since s'' is the number of edges and face diagonals in the whole scene (see Remark 4.8). Therefore, $O(p + s'(\log s' + \log \delta) + s'')$ is in $O(p \log p + s \log s)$, and the theorem follows. □

Note that the complexity of the computation also covers the preprocessing of binary search data structures defined in Remark 3.1.

We finally note that, if desired, the graph exploration method presented above can be applied to a subset of the input polytopes. In this case, we first find all the primary vertices that are related to the polytopes of the subset, and then apply the graph exploration to these primary vertices only.

5. Tightness of the Succinct Skeleton

In this section, we show, mostly by examples, that Theorem 4.9 is tight in the sense that if any one of the primary vertex types *EEEE*, *VEE*, *FEE*, or *VV*, is regarded instead as a secondary vertex type, and thus excluded from the succinct skeleton, then Theorem 4.9 no longer holds.

Type *EEEE*. Any vertex of type *EEEE* requires four support polytopes, and by the general position assumption, there are no skeleton arcs that have four support polytopes. Hence, by Lemma 3.2, vertices of type *EEEE* must be regarded as primary.

Types *VEE* and *FEE*. When three input polytopes are not the support polytopes of any *EEEE* vertex, then the vertices of types *VEE* and *FEE* that have supports on the three polytopes cannot be computed from any *EEEE* vertex.

Moreover, some scenes may generate vertices of type *VEE* but no vertices of type *FEE*, as shown in Figure 13 (a). Hence, by Lemma 3.2, vertices of type *VEE* cannot be dropped.

Furthermore, some scenes may generate vertices of type *FEE* but no vertices of type *VEE*, as shown in Figure 13 (b) and explained below.

The scene in Figure 13 (b) consists of a prism that approximates a cylinder, positioned between two truncated pyramids that approximate truncated cones, where

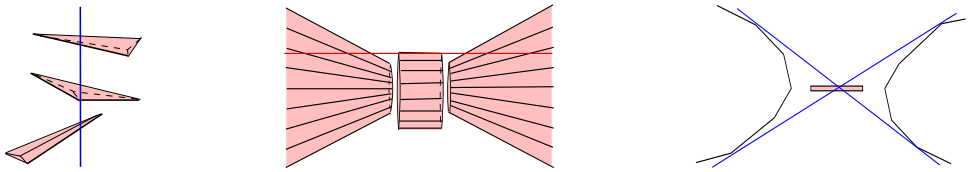


Fig. 13. (a) Three polytopes admit vertices of type *VEE* but not vertices of *FEE*. (b) Three polytopes admit vertices of type *FEE* but not vertices of type *VEE*. (c) A cross section of (b) as indicated by the red line segment.

the full cones would barely intersect.

A supporting plane of a face of the prism intersects the two truncated pyramids in two polygonal arcs that approximate hyperbolas (Figure 13 (c)). These two polygonal arcs admit two bitangents that lie in the supporting plane of the face of the prism and that cross the face, generating two *FEE* vertices.

As the supporting plane rolls around the prism, either one of the two bitangents will cross a pyramid face, or both bitangents will cross a face of the prism. Thus for this scene, each arc of type *EEE* is incident to two *FEE* vertices. Since there are no vertices of type *VEE*, the vertices of type *FEE* cannot be computed from vertices of type *VEE*. Hence, by Lemma 3.2, vertices of type *FEE* cannot be dropped.

Type *VV*. When two input polytopes are not the support polytopes of any *EEEE*, *VEE* or *FEE* vertex, then the vertices of type *VV* that have supports on the two polytopes cannot be computed from any *EEEE*, *VEE* or *FEE* vertex. Moreover, when two polytopes resemble two nearly flat tetrahedra and face each other, then the only primary vertices they admit are of type *VV*. Therefore, vertices of type *VV* must be regarded as primary.

6. Conclusion

We have presented a method to recover a full visibility skeleton, either partial or complete, from a succinct one. The full visibility skeleton is the 0D and 1D cells of the 3D visibility complex of polytopes [10,11], whereas the succinct one is defined by visual event surfaces, and is a subset of the full one [5,7]. Recovering the full skeleton mainly consists of computing the secondary vertices of type *FvE*, *FF*, and *VV* (whose supports do not lie on a plane that is tangent to both support polytopes), from the primary vertices of type *VEE*, *FEE*, and *VV* (whose supports lie on a plane that is tangent to both support polytopes).

Given k polytopes with n edges in total, the running time of our method is, in the worst case, $O(p \log p + s \log s)$, where p is the number of primary vertices except type *EEEE*, and s is the number of secondary vertices. In the worst case, p and s are of size $O(n^2 k)$ and $O(n^2)$ respectively, which gives a worst-case total complexity of $O(n^2 k \log n)$. This worst-case complexity is the same as the best previously known

algorithm, which consists in computing $O(n^2)$ candidate secondary vertices in a brute-force way and checking the occlusion with each of the k polytopes in $O(\log n)$ time each (using the Dobkin-Kirkpatrick hierarchical representation [8,20]). However, we can expect our method to be much more efficient on many instances since its running time is output sensitive.

An interesting subject for future research is to generalize our results to other types of input, such as intersecting polytopes and non-convex polyhedra, and to consider objects in, possibly degenerate, arbitrary positions.

References

[1] P. K. Agarwal and M. Sharir. Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions. *SIAM Journal on Computing*, 25:100–116, 1996.

[2] T. Akenine-Möller and U. Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In Proceedings of the *13th Eurographics Workshop on Rendering (EGRW'02)*, pages 297–306, Aire-la-Ville, Switzerland, 2002. Eurographics Association.

[3] H. Brönnimann, O. Devillers, V. Dujmović, H. Everett, M. Glisse, X. Goaoc, S. Lazard, H.-S. Na, and S. Whitesides. Lines and free line segments tangent to arbitrary three-dimensional convex polyhedra. *SIAM Journal on Computing*, 37(2):522–551, 2007.

[4] H. Brönnimann, H. Everett, S. Lazard, F. Sottile, and S. Whitesides. Transversals to line segments in three-dimensional space. *Discrete and Computational Geometry*, 34(3):381–390, 2005.

[5] J. Demouth. *Événements visuels et limites d'ombres*. PhD thesis, Université Nancy 2, Nov. 2008.

[6] J. Demouth and X. Goaoc. Topological changes in the apparent contour of convex sets, 2008. Manuscript.

[7] J. Demouth and X. Goaoc. Computing direct shadows cast by convex polyhedra. In *Proceedings of the 25nd European Workshop on Computational Geometry*, March 2009.

[8] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: a unified approach. In Proceedings of the *17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 400–413. Springer, 1990.

[9] F. Duguet and G. Drettakis. Robust epsilon visibility. In Proceedings of the *29th annual conference on Computer graphics and interactive techniques (SIGGRAPH'02)*, pages 567–575, New York, NY, USA, 2002. ACM.

[10] F. Durand. *Visibilité tridimensionnelle : étude analytique et applications*. PhD thesis, Université Joseph Fourier - Grenoble I, 1999.

[11] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: a powerful and efficient multi-purpose global visibility tool. In Proceedings of the *24th annual conference on Computer graphics and interactive techniques (SIGGRAPH'97)*, pages 89–100, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[12] F. Durand, G. Drettakis, and C. Puech. Fast and accurate hierarchical radiosity using global visibility. *ACM Transactions on Graphics*, 18(2):128–170, 1999.

[13] F. Durand, G. Drettakis, and C. Puech. The 3D visibility complex. *ACM Transactions on Graphics*, 21(2):176–206, 2002.

[14] T. Funkhouser, N. Tsingos, I. Carlbom, G. Elko, M. Sondhi, J. E. West, G. Pingali, P. Min, and A. Ngan. A beam tracing method for interactive architectural acoustics. *The Journal of the Acoustical Society of America*, 115(2):739–756, 2004.

[15] M. Glisse. *Combinatoire des droites et segments pour la visibilité 3D*. Thèse d'université, Université Nancy 2, Oct 2007.

[16] X. Goaoc. *Structures de visibilité globales : tailles, calculs et dégénérescences*. Thèse d'université, Université Nancy 2, May 2004.

[17] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, Dec. 2003.

[18] T. He, L. Hong, D. Chen, and Z. Liang. Reliable path for virtual endoscopy: ensuring complete examination of human organs. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):333–342, Oct.-Dec. 2001.

[19] J. Koenderink and A. van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, 24:51–59, 1976.

[20] J. O'Rourke. *Computational Geometry in C*. Cambridge Univsersity Press, 2nd edition, 1998.

[21] S. Parker, P. Shirley, and B. Smits. Single sample soft shadows. Technical Report UUCS-98-019, Computer Science Department, University of Utah, 1998.

[22] M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993.

[23] M. Pocchiola and G. Vegter. The visibility complex. *International Journal of Computational Geometry and Applications*, 6(3):279–308, 1996. Proceedings of the 9th ACM Annual Symposium on Computational Geometry (SoCG'93).

[24] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proceedings of the Eurographics Workshop on Rendering (EGRW'00)*, pages 71–82, London, UK, 2000. Springer-Verlag.

[25] L. Zhang, H. Everett, S. Lazard, C. Weibel, and S. Whitesides. On the size of the 3D visibility skeleton: experimental results. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08)*, volume 5193 of *Lecture Notes in Computer Science*, pages 805–816, Karlsruhe, Germany, Sept. 2008. Springer.